Direct X Midterm: Flag Day

Objective

Create a scene with a windy flag using your knowledge of Geometry, Shaders and Texturing.

Exam Instructions

You have 6 hours to complete as much of this exam as possible. You may reference any previously completed labs, as well as slides and any documentation.

Scoring Breakdown

Sky background image is rendered correctly. 15 points

Ground rendered and textured correctly. 10 points

Crates properly positioned and textured. 10 points

Camera is positioned correctly in scene. 5 points

Flag pole mesh renders correctly. 5 points

Flag pole mesh uses normal data for “fake” lighting effect. 10 points

Flag rendered at proper location. 10 points

Flag rendered with flag texture. 10 points

Flag shader rejects pixels with low alpha. 10 points

Flag shader waves flag according to formula. 15 points

**Total 100 points**

Explanation of Tasks

**The Sky:**

The sky is just a **textured quad** that must be rendered **behind** the rest of the scene in **clip space**. Everything else must be rendered on top of this quad.

**The Ground:**

The ground is a quad that ranges from **-10x,0y,10z to 10x,0y,-10z**. You must **add UV coordinates** and display the ground texture across the surface. The texture must **WRAP** across the surface **3 times**.

**The Crates:**

The crates are **cubes** that are **built around the origin** that **span 1.5 units across on each axis**. You will need to correctly **add UV coordinates** to the cube data to display the crate texture on the surface of each cube.

The 3 crates are placed in the following locations:

crate1 [**5x,0.75y,-3z**] crate2 **[-3x,0.75y,-2z**] crate3 [**1x,0.75y,4z**]

**The Flag Pole:**

***The Pole:***

The flag's pole is a 3D mesh whose data is stored inside “flagpole.h”. You must transfer this data to VRAM much like any other handmade piece of geometry. The mesh includes normal data which you will utilize to create the “fake” lighting effect in a shader. At this point you have not dealt with normal data and do not know how to do lighting. So for this exam I want you to **think of the normal data as color data** instead (minus the alpha).

***The “Fake” Lighting:***

Create a vertex & pixel shader that will use your position & “color” information. The position data should be treated like any other 3D object and sent to the rasterizer. The “color” data should be sent to the pixel shader so we can use it to shade the surface.

In the pixel shader use the following formula to create the shading effect:

**Brightness = NormalX + NormalY**

**OutColorRGBA = Brightness, Brightness, Brightness, 1.0**

**The Flag:**

***Geometry:***

The flag is a vertical plane of **25x25** vertices that **span** from **0x,3y,0z to 5x,0y,0z**. Each vertex is spaced **apart from each other based on the distance needed to cover the span**. Besides position, each vertex contains a **UV coordinate as well**. To render these vertices as a **list of triangles**, you must build an **index list** to form all the needed triangles. The following algorithms can be used to assist you when **building the index array**.

**numTriangles = (TotalGridWidth -1) \* (TotalGridLength -1) \* 2**

(the below formula might be used during a nested loop, forming your triangles)

**vertexStartIndex = currGridRow \* GridWidth + currGridColumn**

There are **multiple ways to form the vertex and index buffers you will need**, but the algorithms above can be used to **assist** you when attempting to figure out how many triangles you will need, and how to convert a 2D grid location into a 1D vertex index location. As for **UV's** you will need to have them **span across the surface** of the plane to show the entire texture.

***Positioning the Flag:***

Since the flag is built with its bottom left corner **sitting at the origin**, we will need to **translate it 4.9 units along the Y** in order to position it correctly on the flag pole.

***Waving the Flag:***

The flag is a custom shader that textures the plane with the **"finnishflag.dds"** texture and uses the **following formulas** in the **vertex shader** to create the waving effect.

**worldVertPosZ += cosine( worldVertPosX + TIME \* 5 ) \* worldVertPosX \* 0.1**

**worldVertPosZ += cosine( worldVertPosY + TIME \* 7 ) \* worldVertPosX \* -0.1**

***Showing the rips in the flag:***

To create the holes in the flag, you’ll need to check the **alpha value of the texture** and **discard** if the alpha value is **less than 0.85**.

**Misc Values:**

***Camera Location:*** Eye[**5x,2y,-9z**] LookAt[**1x,3y,0z**] Up[**0x,1y,0z**]

***Projection Values:*** Field of View[**75degrees**] Znear[**0.1**] Zfar[**100**]

***Screen Resolution:*** [**1280 x 768**]

**Hints & Tips:**

Clip space is explained near the end of day 2.

A vertex buffer need not store only local space data… also keep in mind XYZ / 1.0 = XYZ.

Rendering a clip space quad may pollute the contents of your Z buffer.

If you can figure out what the starting vertex for your indexed triangle is, the others are close by.

Removing pixels based on alpha can be done from within the pixel shader itself. (Look over the MSDN list of shader intrinsics)

**Turn In:**

Please be sure your project compiles. **A project that has compiler errors is a ZERO.**

**Take a screenshot of your exam and include it with your turn-in.(printScreen/Fraps)**

Delete your Debug/Release/ipch folders, zip and turn in as **LastName.FirstName.Midterm.zip**

**When you are done, please confirm your file was turned in and then you may leave.**